Programação Visual
Trabalho de Laboratório nº 6

Objetivo	MVC – Introdução. Aplicações básicas com acesso a base de dados através do Entity Framework e utilização do pacote Microsoft Identity.
Programa	Pretende-se criar um Site de compras de livros. A aplicação deverá permitir aos utilizadores visualizar os livros, onde as lojas se situam e adicionar livros ao carrinho de compras. Deverá existir um administrador que faz a gestão de stock dos livros.
Regras	Implementar o código necessário e testar no fim de cada nível. Use as convenções de codificação adotadas para a linguagem C# e para o modelo MVC.

Nível 1

- Crie uma aplicação **ASP.NET Core Web Application (Model-View-Controller)**, com o o nome de ESTSBooks e escolha como método de autenticação Individual Accounts.
- Na pasta Models, crie os seguintes modelos:
  - **BookCategory** – Enumerado com os valores “Action”, “Romance”, “Triller”, “Comedy”.
  - **Book** – BookId (Guid), Title (string), Price (decimal), Units (int), Category (BookCategoryEnum), BookStoreId (Guid), BookStore (BookStore?) - e**sta propriedade deve ser nullable.**
  - **BookStore** – BookStoreId (Guid), Location (string), Books (List<Book>) - **inicialize esta propriedade como uma lista vazia**
- Na classe Book, acrescente as seguintes anotations nas respetivas propriedades:
  - BookId: `[Key]`
  - Title: `[MaxLength(150)]`
  - Category: `[EnumDataType(typeof(BookCategory))]`
  - Price: `[Column(TypeName="decimal(10, 2)"), DataType(DataType.Currency)]`
  - `Units: [Range(0, 999)]`
  - BookStoreId: `[ForeignKey("BookStoreId")]`
- Na classe BookStore, acrescente a annotation [Key] à propriedade BookStoreId.
- Usando scaffolding, gere os controllers com views para os Livros e as Lojas, **certifique-se de que usa o Entity Framework, e de que seleciona o Data Context Class.**
- Adicione à barra de menus os links para as views Index dos controladores que criou.
- Abra a aba “Package Manager Console” no Visual Studio e corra os seguintes comandos:

```
Add-Migration Initial
Update-Database
```

- Verifique que a aplicação compila e não tem nenhum erro (não precisa de correr a aplicação).

Programação Visual
Trabalho de Laboratório nº 6

Nível 2

- Na view Books/Create acrescente ao `select` das categorias o attributo html `asp-items="ViewBag.Categories".`
- No BooksController, na ação do Create (GET), adicione os elementos do enumerado `BookCategory` ao `ViewData["Categories"]`.
    - Itere sobre os itens do enumerado, e crie uma projeção de um objeto anónimo com duas propriedades: Value (int) e Name (string).
- Repita este processo para as restantes views e ações referentes aos livros.
- Na classe ApplicationDbContext (dentro da pasta Data) acrescente o seguinte método:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
   base.OnModelCreating(modelBuilder);
}
```

- No método acima, adicione as linhas necessárias para instanciar 5 Livros e 2 Lojas. Depois faça Data Seeding utilizando os objetos criados - para ver mais sobre data seeding com EF Core, visite a documentação.

Nota: Irá necessitar de "reiniciar" a base de dados, para tal, abra o "SQL Server Object Explorer" e apague a base de dados. Remova a pasta "Migrations" e os ficheiros dentro desta. Volte a correr os comandos anteriores para criar uma migração e atualizar a base de dados.

A partir de agora, ao criar os modelos, o método desenvolvido irá ser executado e criará os dados.

Programação Visual
Trabalho de Laboratório nº 6

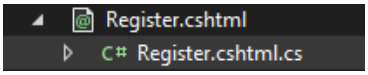
Nível 3

- Na pasta Models, crie uma nova class BookUser que estende IdentityUser. Adicione uma propriedade Nome (string) com a Data Annotation `[PersonalData]`.
- Altere a classe ApplicationDbContext de forma a esta estender `IdentityDbContext<BookUser>`.
- Adicione um novo item de Scaffolding ao projeto e selecione "Identity", escolhendo depois o override dos ficheiros:

  "Account\Register", "Account\Login", "Account\Logout", "Account\Manage\index"

- Use o contexto da aplicação no campo "Data Context Class".

Na pasta "Areas" irá encontrar algumas views geradas relativas aos utilizadores. Estas views tem um PageModel que vamos ter de editar:



- Para view Index, abra o seu PageModel e altere a class InputModel (no mesmo ficheiro), acrescentado a seguinte propriedade e as suas *data annotations*:

```
[Required]
[DataType(DataType.Text)]
[Display(Name = "Full name")]
public string Name { get; set; }
```

- Altere ainda o metodo LoadAsync e acrescente a nova propriedade na inicialização do InputModel.
- Para o Page Model da view Register, altere da mesma forma o InputModel, e no metodo OnPostAsync garanta que atribui o valor à propriedade Name logo após a chamada de CreateUser(). Apague a propriedade `_emailSender` e remova o código que a utilizava.
- Acrescente ainda o campo nome na View do Register, seguindo o padrão já existente de forma ao campo ser validado.
- Na pasta das View/Shared, na view LoginPartial, atualize as linhas referentes à injeção dos Managers, para usarem BookUser em vez de IdentityUser.
- No ficheiro Program.cs, substitua IdentityUser por BookUser, e mude a opção RequireConfirmedAccount para false.

Nota: Caso tenha dúvidas pode usar esta página da documentação para se guiar.

- Para testar, **"reinicie" novamente a base de dados,** e tente fazer o registo e o login de um utilizador.

Programação Visual
Trabalho de Laboratório nº 6

Nível 4

- Crie uma nova classe estática no projeto "Configurations", dentro desta adicione o seguinte método:

```
public static async Task CreateRoles(IServiceProvider serviceProvider)
{
  var roleManager = serviceProvider.GetRequiredService<RoleManager<IdentityRole>>();
  var userManager = serviceProvider.GetRequiredService<UserManager<BookUser>>();
  string[] roleNames = { "Manager", "Default" };

  foreach (var roleName in roleNames)
  {
    var roleExist = await roleManager.RoleExistsAsync(roleName);
    if (!roleExist) await roleManager.CreateAsync(new IdentityRole(roleName));
  }

  var manager = new BookUser
  {
    Name = "Manager Account",
    UserName = "manager@ips.pt",
    Email = "manager@ips.pt"
  };

  var _user = await userManager.FindByEmailAsync(manager.Email);
  if (_user != null) return;

  var createPowerUser = await userManager.CreateAsync(manager, "Password_123");
  if (createPowerUser.Succeeded)
    await userManager.AddToRoleAsync(manager, "Manager");
}
```

- No ficheiro Program.cs, altere a linha onde chama "AddDefaultdentity" para o seguinte:

```
builder.Services.AddIdentity<BookUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>()
        .AddDefaultTokenProviders();
builder.Services.AddRazorPages();
```

- Ainda no mesmo ficheiro, adicione as seguintes linhas antes da linha "app.Run()":

```
using var scope = app.Services.CreateScope();
await Configurations.CreateRoles(scope.ServiceProvider);
```

Com este novo código, estamos a criar os nosso Roles, que iremos usar para filtrar o acesso a diferentes partes do sistema, estamos ainda a criar um novo utilizador com o Role de Manager (sendo que a nossa página de registo não permite escolher o Role a atribuir a um utilizador).

- Volte a reiniciar a base de dados.
- Registe novamente um utilizador para si e experimente fazer login e logout com o seu utilizador e também com o utilizador "Manager" (username: manager@ips.pt, password: Password_123).

Programação Visual
Trabalho de Laboratório nº 6

Nível 5

- Vamos agora proteger certas operações, de forma que apenas utilizadores com o Role Manager consigam aceder.
- No BookStoresController, acrescente a anotação `[Authorize(Roles = "Manager")]` ao nível do controller.
- No BooksController, acrescente a anotação `[Authorize(Roles = "Manager")]` às ações de Criar, Edit e Apagar (GET e POST).
- Na view do menu principal, rodei o elemento do BookStore numa validação utilizado o metodo `User.IsInRole`, de forma que apenas utilizadores com o Role Manager consigam ver o item no menu.
- Utilize novamente o método `User.IsInRole` para esconder das views os links para Criar, Editar e Apagar da view Index dos livros.

Desafio

- Crie um novo Role "Supplier", e acrescente as validações necessárias para que este possa apenas Listar e Criar livros.

Notas

Para os identificadores siga as convenções adotadas pelo C#, nomeadamente:

- A notação camelCase para o nome das variáveis locais e identificadores privados.
- A notação PascalCase para os nomes públicos dos métodos e classes
- Não utilize o símbolo '_' nos identificadores
- Não use abreviaturas